Phase 2 review

- Identify a variable $x$ that increases the value of the objective function.
- Identify an equality $y = \ldots$ that most severely restricts how much $x$ can be increased.
-Swap $x$ and $y$: make $x$ a basic variable.
- Keep repeating this until:
  - Objective value cannot be increased.
  - It can be increased by increasing $x$, but there is no bound on how much $x$ can increase.

Overview of Phase I

- Phase 0: Express the linear program in slack form.
- Phase I: Compute a feasible basic solution, if one exists.
- Phase II: Optimize the value of the objective function.

Constructing the Auxiliary Linear Program

Consider the original LP problem:

Maximize $\sum_{j=1}^{n} c_j x_j$
Subject to:

$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad (i \in [1, m])$
$x_j \geq 0 \quad (j \in [0, n])$

This problem is feasible if and only if the optimal value for $L_{aux}$ is zero.

\[ \text{Maximize} \quad -x_0 \]
\[ \text{Subject to:} \]

$\sum_{j=1}^{n} a_{ij} x_j - x_0 \leq b_i \quad (i \in [1, m])$
$x_j \geq 0 \quad (j \in [0, n])$

Justification for Auxiliary LP

Maximize $-x_0$
Subject to:

$\sum_{j=1}^{n} a_{ij} x_j - x_0 \leq b_i \quad (i \in [1, m])$
$x_j \geq 0 \quad (j \in [0, n])$

$\Rightarrow$ Suppose $x_0$ has optimal value 0. Then clearly $a_{ij} x_j \leq b_i$ is satisfied for all inequalities.

$\Leftarrow (a)$ Suppose original problem has feasible solution $x^\ast$. Then $x^\ast$
combined with $x_0 = 0$ is feasible solution for $L_{aux}$.

$\Leftarrow (b)$ Due to the non-negativity constraint, $-x_0$ can be at most 0; thus, this solution is optimal for $L_{aux}$. 
Finding Feasible Basic Solution for $L_{aux}$

- So far, we argued that original problem $L$ has feasible solution if $L_{aux}$ has optimal value 0.
- But we still need to figure out how to find feasible basic solution to $L_{aux}$.
- Next: We’ll see how we can find feasible basic solution for $L_{aux}$ after one pivot operation.

Why is This True?

- Suppose this equality has most negative $b_i$:
  \[ x_i = b_i + x_0 - \sum_{j=1}^{n} a_{ij} x_j \]
- Rewrite to make $x_0$ basic:
  \[ x_0 = -b_i - x_i + \sum_{j=1}^{n} a_{ij} x_j \]
- Now, $-b_i$ is positive and greater than all other $|b_j|$'s
- Thus, when we plug in equality for $x_0$ into other equations, their new constants will be positive
- Hence, we find a feasible basic solution after at most one pivot step

Auxiliary Problem in Slack Form

\[ z = -x_0 \]
\[ x_i = b_i + x_0 - \sum_{j=1}^{n} a_{ij} x_j \]

- If all $b_i$'s are positive, basic solution already feasible
- If there is at least some negative $b_i$, find equality $x_i$ with most negative $b_i$
- Make $x_0$ new basic variable, and $x_i$ non-basic
- Claim: After this one pivot operation, all $b_i$'s are non-negative; thus basic solution is feasible

Example

- Consider the following linear program from earlier:
  \[
  \begin{align*}
  z &= 2x_1 - x_2 \\
  x_3 &= 2 - 2x_1 + x_2 \\
  x_4 &= -4 - x_1 + 5x_2 
  \end{align*}
  \]
- Construct $L_{aux}$:
  \[
  \begin{align*}
  z &= -x_0 \\
  x_3 &= 2 + x_0 - 2x_1 + x_2 \\
  x_4 &= -4 + x_0 - x_1 + 5x_2 
  \end{align*}
  \]
- Which equation has most negative constant?
- Swap $x_4$ and $x_0$:
  \[ x_0 = 4 + x_4 + x_1 - 5x_2 \]

Example, cont

- After pivoting, we obtain the new slack form:
  \[
  \begin{align*}
  z &= -4 - x_4 - x_1 + 5x_2 \\
  x_3 &= 6 - x_1 - 4x_2 + x_4 \\
  x_0 &= 4 + x_4 + x_1 - 5x_2 
  \end{align*}
  \]
- What is current objective value?
- How can we increase it?
- Which equation constrains $x_2$ the most?
- Swap $x_2$ and $x_0$:
  \[ x_2 = \frac{4}{5} - \frac{1}{5} x_0 + x_4 + x_1 \]

Example, cont

- After pivoting, new slack form:
  \[
  \begin{align*}
  z &= -x_0 \\
  x_2 &= \frac{4}{5} - \frac{4x_1}{5} + \frac{9x_4}{5} \\
  x_3 &= \frac{1}{5} + \frac{1x_1}{5} - \frac{2x_4}{5} + \frac{9}{5} x_2 
  \end{align*}
  \]
- Objective function cannot be increased, so we are done!
- In original problem, objective function was $z = 2x_1 - x_2$
- Since $x_3$ is now a basic variable, substitute for $x_2$ with RHS:
  \[ z = -4 + \frac{9x_1}{5} - \frac{x_4}{5} \]
- Thus, Phase I returns the following slack form to Phase II:
  \[
  \begin{align*}
  z &= \frac{1}{5} + \frac{1}{5}x_0 - \frac{3}{5} \\
  x_2 &= \frac{4}{5} - \frac{4x_1}{5} + \frac{9}{5} \\
  x_3 &= \frac{1}{5} + \frac{1x_1}{5} - \frac{2x_4}{5} + \frac{9}{5} x_2 
  \end{align*}
  \]
**Overview of Techniques**

- Two different techniques for solving linear integer inequalities
  1. Elimination-based techniques: Omega Test, Cooper’s method
  2. Relaxation-based techniques: Branch-and-bound, Gomory cuts, Cuts-from-Proofs

**Problem Description**

- As in previous two lectures, we’ll consider $T_Z$ formulas without disjunctions
- **Problem we want to solve**: Given an $m \times n$ matrix $A$ with only integer coefficients and a vector $\vec{b}$ in $\mathbb{Z}^n$, does

$$A\vec{x} \leq \vec{b}$$

have any integer solutions?
- Integrality requirement actually makes problem much harder
- Finding solution over rationals is poly-time, but integer problem is NP-complete even without disjunctions

**Theory of Integers**

- Earlier, we talked about the theory of integers $T_Z$
- **Signature of $T_Z$**:
  $$\Sigma_Z : \{\ldots, -2, -1, 0, 1, 2, \ldots, -3, -2, 2, 3, \ldots, +, -, =, >\}$$
- This theory also called linear arithmetic over integers
- Since equal in expressive power to Presburger arithmetic, people also refer to it as Presburger arithmetic
- Today and next lecture: Look at algorithms for deciding satisfiability in quantifier-free fragment of $T_Z$

**A Concrete Example**

- Consider the set of linear inequalities:

$$
3x + 3y \leq 2 \\
3x + 3y \geq 1
$$

- This problem has rational-valued solutions, e.g., $x = \frac{1}{3}, y = \frac{1}{4}$
- But it doesn’t have integer solutions
- In general, if $A\vec{x} \leq \vec{b}$ has integer solutions, it also has rational solutions
- But if it has rational solutions, this does not imply it also has integer solutions

**Road Map**

- Today’s lecture:
  Talk about an elimination-based technique called Omega test
- Next lecture:
  Talk about two relaxation-based techniques:
  1. Branch-and-Bound
  2. Cuts-from-Proofs

**Summary**

- To solve constraints in $T_Q$ (linear inequalities over rationals), we use Simplex algorithm for LP
- Simplex has two phases
- In first phase, we construct slack form such that it has a basic feasible solution
- In second phase, we start with basic feasible solution and rewrite one slack form into equivalent one until objective value can’t increase

**Theory of Integers**

- Earlier, we talked about the theory of integers $T_Z$
- **Signature of $T_Z$**:
  $$\Sigma_Z : \{\ldots, -2, -1, 0, 1, 2, \ldots, -3, -2, 2, 3, \ldots, +, -, =, >\}$$
- This theory also called linear arithmetic over integers
- Since equal in expressive power to Presburger arithmetic, people also refer to it as Presburger arithmetic
- Today and next lecture: Look at algorithms for deciding satisfiability in quantifier-free fragment of $T_Z$
The Omega Test: Historical Perspective

- Omega Test: invented in early 1990's for compiler optimizations
- Particular application: array dependence analysis
- Array dependence analysis: “Can two expressions $a[i]$ and $a[j]$ refer to the same element?”
- Can use this information to reorder reads and writes from the array and perform operations in parallel

Array Dependence Analysis Example

- Consider the following code snippet:
  ```c
  for(i=1; i<= 100; i++) {
    for(j=i; j<= 100; j++)
      a[i, j+1] = a[100, j]
  }
  ```

  - Can the expressions $a[i, j+1]$ and $a[100, j]$ ever refer to the same element (not necessarily in the same iteration)? No!
  - Thus, no array element is both read and written to in the loop
  - Hence, we can optimize code by performing assignments in parallel!

Array Dependence Analysis as Integer Constraints

```c
for(i=1; i<= 100; i++) {
  for(j=1; j<= 100; j++)
    a[i, j+1] = a[100, j]
}
```

- Can express dependence analysis as linear integer constraints
- Variables $w_i$ and $w_j$ denote array indices when write is performed
- Variables $r_i$ and $r_j$ denote array indices when read is performed
- How do we express that same element is both read and written to? $w_i = r_i \land w_j = r_j$

Applications of Theory of Integers

- Array dependence analysis one application of decision procedure for theory of integers
- Omega Test was initially invented to do a better job with array dependence analysis
- Many other applications in software verification, compiler optimizations, operations research, …
Omega Test: Main Idea

- **Main idea**: Eliminate variables one by one from the initial system $A\vec{x} \leq \vec{b}$

- Geometrically, eliminating a variable corresponds to computing a projection of a polytope in $n$-dimensional space to an $n-1$-dimensional space

- Since the polytope has one less dimension at each step, resulting problem is easier to solve than the previous one

Fourier-Motzkin Variable Elimination

- Suppose we want to eliminate variable $x_n$ from $A\vec{x} \leq \vec{b}$

- Consider an inequality $\sum_{j=1}^{n} a_{nj} x_j \leq b_i$

- This can be rewritten as $a_{ni} x_n \leq b_i - \sum_{j=1}^{n-1} a_{nj} x_j$

- If $a_{ni}$ is positive, this yields an upper bound on $x_n$:

  $$x_n \leq \frac{b_i}{a_{ni}} - \sum_{j=1}^{n-1} \frac{a_{nj}}{a_{ni}} x_j$$

- If $a_{ni}$ is negative, this yields a lower bound on $x_n$:

  $$x_n \geq \frac{b_i}{a_{ni}} - \sum_{j=1}^{n-1} \frac{a_{nj}}{a_{ni}} x_j$$

Projections in Omega Test

Omega test computes three kinds of projections, called shadows:

1. **Real Shadow**
   - Overapproximates satisfiability over integers
   - If real shadow has no solutions, neither does original problem

2. **Dark Shadow**
   - Underapproximates satisfiability over integers
   - If dark shadow has solution, original problem has solution

3. **Gray Shadows**
   - These correspond to areas between real and dark shadow that might contain integer points
   - Omega test constructs multiple gray shadows

The Real Shadow

- When constructing the real shadow, we ignore requirement that solution must be integer

- Thus, resulting projection overapproximates satisfiability of original problem

- To construct real shadow, we use the Fourier-Motzkin variable elimination technique

Fourier-Motzkin Variable Elimination, cont.

- Thus, if we have $A\vec{x} \leq \vec{b}$ has two rows $i$ and $k$ with positive and negative coefficients for $x_n$, this yields the inequality:

  $$\frac{b_i}{a_{kn}} - \sum_{j=1}^{n-1} \frac{a_{kj}}{a_{kn}} x_j \leq x_n \leq \frac{b_k}{a_{kn}} - \sum_{j=1}^{n-1} \frac{a_{kj}}{a_{kn}} x_j$$

- We eliminate $x_n$ by removing it from the middle of inequality:

  $$\frac{b_i}{a_{kn}} - \sum_{j=1}^{n-1} \frac{a_{kj}}{a_{kn}} x_j \leq \frac{b_i}{a_{kn}} - \sum_{j=1}^{n-1} \frac{a_{kj}}{a_{kn}} x_j$$

- If we do this for every pair of inequalities with positive and negative coefficients for $x_n$, this yields the real shadow
### Fourier-Motzkin Example

- Consider the set of inequalities:
  \[ x \leq y + 10 \quad y \leq 15 \quad -x + 20 \leq y \]
- Let's compute real shadow on \( x \)-axis using Fourier-Motzkin
- Isolate \( y \) on one side:
  \[ (1) \ x - 10 \leq y \quad (2) \ y \leq 15 \quad (3) \ -x + 20 \leq y \]
- From (1) and (2), we get \( x - 10 \leq 15 \), i.e., \( x \leq 25 \)
- From (2) and (3), we get \( -x + 20 \leq 15 \), i.e. \( x \geq 5 \)
- Thus, real shadow on \( x \)-axis is \( 5 \leq x \leq 25 \)

### Dark Shadow

- The second projection Omega test constructs is dark shadow
- Dark shadow underapproximates satisfiability
- Suppose we want to eliminate variable \( x \) from \( A\vec{x} \leq \vec{b} \)
- Dark shadow only projects those parts of polytope that are at least one unit thick in the \( x \)-dimension
- If dark shadow has integer solution, original polytope must also have integer solution. Why?
- Since polytope is at least one unit thick above the dark shadow in \( x \)-dimension, we are guaranteed to have an integer solution for \( x \) as well!

### Real Shadow and Overapproximation

- When we want integer solutions, real shadow overapproximates satisfiability
- For instance, consider \( 3x \geq 1 \land 3x \leq 2 \)
- Does this formula have integer solutions? no
- When we compute dark shadow, we get \( \frac{1}{3} \leq \frac{3}{2} \), i.e., \( \frac{1}{2} \leq 2 \)
- Since this formula is tautology, real shadow is satisfiable
- But original formula is not satisfiable

### Math Behind the Dark Shadow

- As in real shadow, consider a pair of inequalities corresponding to lower and upper bounds on \( x \):
  \[ \mathcal{L} \leq ax \quad bx \leq \mathcal{U} \]
- These imply:
  \[ \frac{\mathcal{L}}{a} \leq x \leq \frac{\mathcal{U}}{b} \]
- Now, suppose there is no integer between \( \frac{\mathcal{L}}{a} \) and \( \frac{\mathcal{U}}{b} \)
- Consider first integer \( i \) smaller than \( \frac{\mathcal{L}}{a} \)

### Math Behind the Dark Shadow, cont.

\[
\begin{align*}
\geq \frac{1}{a} & \quad \geq \frac{1}{b} \\
i & \quad \frac{c}{a} & \quad \frac{d}{b} & \quad i + 1
\end{align*}
\]

- Thus, we have the following inequalities:
  \[ \frac{c}{a} - i \geq \frac{1}{a} \]
  \[ i + 1 - \frac{d}{b} \geq \frac{1}{b} \]
- If we sum these up, we get:
  \[ \frac{c}{a} - \frac{d}{b} + 1 \geq \frac{1}{a} + \frac{1}{b} \]

### Math Behind Dark Shadow, cont

- If we rearrange this equation, we get:
  \[ b\mathcal{L} - ad \geq b + a - ab \]
- Finally, multiplying both sides by \( -1 \):
  \[ a\mathcal{L} - b\mathcal{U} \leq ab - a - b \quad (*) \]
- **Recall:** We derived this equation by assuming that there is no integer solution for \( x \)
- That is, we showed "If there is no integer solution for \( x \), then \((*)\) must hold"
- Thus, negation of \((*)\) guarantees there exists integer solution for \( x \)!
### Math Behind Dark Shadow, cont

- Thus, negation of (*)
  \[ aL - bL > ab - a - b \quad (**) \]
  guarantees there is an integer value for \( x \)!
- Thus, to construct dark shadow, we remove inequalities containing \( x \) and add inequality (**)
- Resulting projection is underapproximation because only projects those parts that are at least one unit thick, but there might be an integer solution for \( x \) even if it’s not unit thick

### Gray Shadows

- Recall: Real shadow overapproximates the problem, and dark shadow underapproximates it.
- If real shadow has integer solutions, but dark shadow does not, we still don’t know if original problem has integer solutions.
- In this case, Omega test constructs projections called gray shadows
- Gray shadows look for integer solutions outside the dark shadow, but inside the real shadow.

### Constructing the Gray Shadow

- Consider again the pair of inequalities:
  \[ L \leq ax \quad bx \leq U \]
- By construction, any point in the real shadow satisfies:
  \[ bL \leq abx \leq alL \quad (1) \]
- Also, by construction, any point outside dark shadow satisfies:
  \[ alL - bL \leq ab - a - b \]
- We can rewrite above as: \( alL \leq bL + ab - a - b \quad (2) \)
- Combining (1) and (2), we have:
  \[ bL \leq abx \leq bL + ab - a - b \]

### Constructing Gray Shadow, cont.

- Thus, any point inside real shadow but outside dark shadow must satisfy:
  \[ bL \leq abx \leq ab + bL - a - b \]
- Dividing by \( b \), points in the gray shadow must satisfy:
  \[ L \leq ax \leq L + \frac{ab - a - b}{b} \]
- Observe: If \( x \) is an integer, \( ax \) must also be integer
- Furthermore, \( ax \) must be equal to
  \[ L + i \]
  for some \( i \) in the range \( (0, \frac{ab - a - b}{b}) \)

### Constructing Gray Shadow, cont.

- Thus, we construct each gray shadow by adding the equality:
  \[ ax = L + i \]
  for each \( i \) in the range \( (0, \frac{ab - a - b}{b}) \)
- If any subproblem has integer solution, then so does original problem
- If no subproblem has integer solution, original problem unsatisfiable

### Remark about Gray Shadows

- Observe: If there are \( n \) integers between 0 and \( \frac{ab - a - b}{b} \), Omega test constructs \( n \) gray shadows
- Thus, Omega test is very sensitive to coefficients in formula
- The larger \( a \) is, the more gray shadows we must consider
- Nightmare for Omega test: Real shadow has solution, dark shadow has no solution, and coefficient \( a \) is very large, and problem is unsatisfiable
- In this case, Omega test must solve a very large number of subproblems
An Optimization

- Omega test uses important optimization to handle equality constraints
- Equality constraints can be expressed as pair of inequalities, but handling equalities directly much more efficient
- Thus, Omega test has special preliminary step where it gets rid of all equality constraints
- Uses interesting coefficient-reducing technique based on symmetric modulo
- Details are in paper posted on class webpage - strongly encouraged to read!

Omega Test Summary

- Omega test is an elimination-based technique for solving linear inequalities over integers
- Constructs three kinds of projections: real shadow, dark shadow, gray shadow
- Problem has no solution if real shadow has no solution
- Problem has solution if dark shadow has solution
- Otherwise, problem has solution iff one of the dark shadows has solution
- Omega test handles equalities specially using the symmetric modulo technique